Group 4: Final Report

Abdel Zaro, Beni Issler and Pedro Monteleone

Opportunity

We aim to address a market gap in accessible and affordable smart medication management systems. Current solutions tend to be large and expensive, or lack smart electronic and mechanical features. We see an opportunity to create a cheap and compact pill manager that ensures convenience, consistency, and safety for individuals taking multiple pills, while enabling real-time monitoring by doctors, caretakers, and family members.

High-level Strategy for Execution

Our high level strategy for addressing this opportunity further limits the scope of the project to a demonstration of the mechanical elements of the system. The proposed IoT elements such as live storage condition monitoring and scheduled pill releases were not prioritized due to the added complexity of coding up an application. However, the device should have the mechanical capacity to integrate these features in the future.

Our device consists of a small square box with 3 containers, for different medications. Each container has a spring loaded lid, which is normally kept shut by a disk in the center of the box. The disk has a cutout slot, and can rotate via a brushed DC motor and gearbox. Upon each button pressed by the user, the disk will spin and align with the next container lid, allowing it to open from the springed hinge. After the next button press, the central disk keeps spinning to align with the next container, while also pushing down the lid that is currently open. The device also contains a temperature and humidity sensor which triggers a buzzing alarm in case the device is under inadequate storage conditions for the pills inside, alerting the user to move the device or store the medicine elsewhere.

Our final prototype fulfilled most of the initial functionality requirements we desired: there are 3 compartments for medicine and one quadrant reserved for electronics; there is one degree of freedom with the rotating disk; the lids are spring loaded and open automatically upon alignment; the rotating disk closes the lids automatically; most importantly, we achieved reliable opening and closing of specific compartments from rest position (with <1% jams or failures). However, there are some more specific goals that we deviated from: Instead of spinning the disk at 2 rpm, as planned, we increased the speed to around 10 rpm to avoid an unnecessarily long opening time; instead of using an absolute encoder, we achieved precise positioning of disk with a fixed limit switch and a incremental optical encoder. These changes were made for practical reasons, and resulted in the same functionality.

Function-Critical Decisions

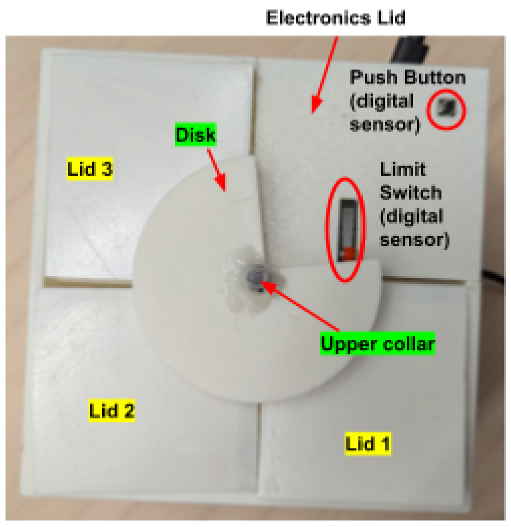
From our CAD model, we know the volume of each individual lid to be 20cm^2. The lid is printed in PLA plastic, at an infill of 20% and density of 1.25 g/cm^3. So, the approximate mass of each lid is $m \approx 20 \cdot 1.25 \cdot 0.2 = 5g$. The distance of the center of mass of the lid to the axis of rotation of its hinge is 34.6mm. So, the minimum torque required to lift the lid is roughly $T_{min} = mgd \approx 0.005kg \cdot 9.81\frac{m}{s^2} \cdot 0.0346m \approx 0.0017Nm \approx 0.015in \cdot lbs$. The spring from McMaster with the lowest available torque that fit the required dimensions of our design provides $0.1in \cdot lbs$, more than 6x the calculated minimum torque. We also purchased a set of $0.25in \cdot lbs$ springs, and after some practical testing decided to use two of the heavier springs to hold the lids, which resulted in a reliable and snappy feel. This means our springs could in theory support 33x the torque from the weight of the lid, although the provided numbers are a maximum torque rating which may not accurately represent the real properties (especially given the low 90° flexion that the springs were subjected to)

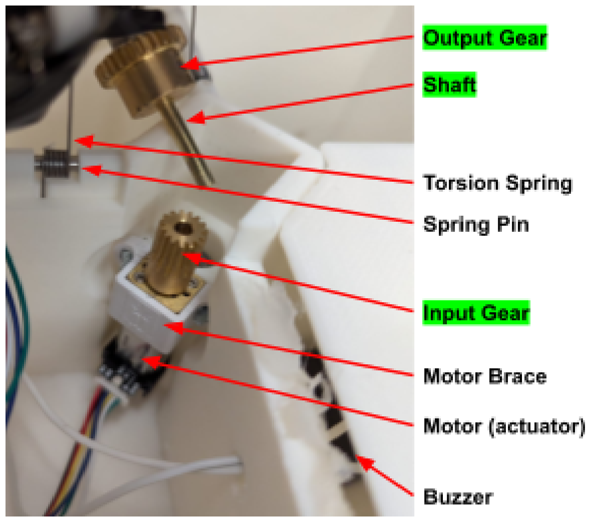
To close the lids, we are relying on torque transferred from the motor to the shaft to the disk, and finally to the lids. The distance between the center plane of the 40mm disk to the axis of rotation of the lids is roughly 9mm. At the very least, the disk must be able to overcome the torque from the springs of one lid, so
$T_{disk,min} \approx r_{disk} F_{disk} \approx r_{disk} F_{spring} \approx 2\frac{r_{disk}}{d_{spring}} T_{spring} \approx 2 * \frac{40}{9} * 0.25 \approx 2.2 in \cdot lbs$. With a reducing gearing ratio of 2, this means the motor should only need to output
$T_{motor,min} \approx \frac{T_{disk,min}}{2} \approx 1.1 in \cdot lbs \approx 1.3 kg \cdot cm$. This does not consider any additional friction anywhere else in the mechanism (which is significant), disturbances like someone touching or holding the lids, or an additional margin for reliability. Since we wanted to drive the motor at low speeds and small size was a priority, we chose a 1000:1 6V Micro Metal Gearmotor HP 6V, with a torque at max efficiency of $2 kg \cdot cm$ and stall torque of $12 kg \cdot cm$.

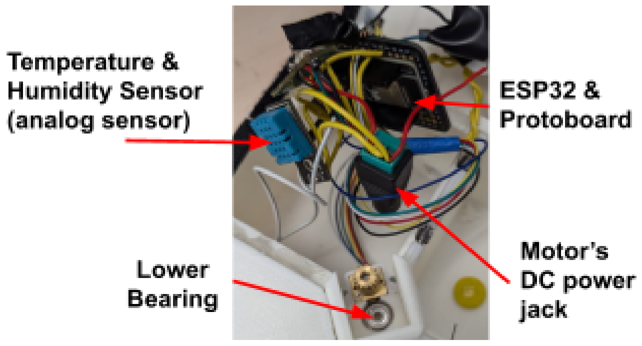
The appropriately sized ball bearings for our device were rated for 40lbs of static load, which is much higher than any reasonable load we expect during operation. So, we found it unnecessary to compute expected shaft, bearing or gear loads, or conduct any FEA analysis to consider deformation or failure of the plastic enclosure.

Labeled Device Images

Below are 3 labeled images of our assembled device. The parts highlighted in green rotate with the main shaft, actuated by the motor. The parts highlighted in yellow rotate around the spring pins, actuated by the torsion springs. The digital and analog sensors used are also identified. Not pictured are the upper bearing, which is nested in the electronics lid (see CAD images in Appendix), the lower collar, which sits below the upper bearing, and the Belleville washers which sit on either side of the upper bearing.

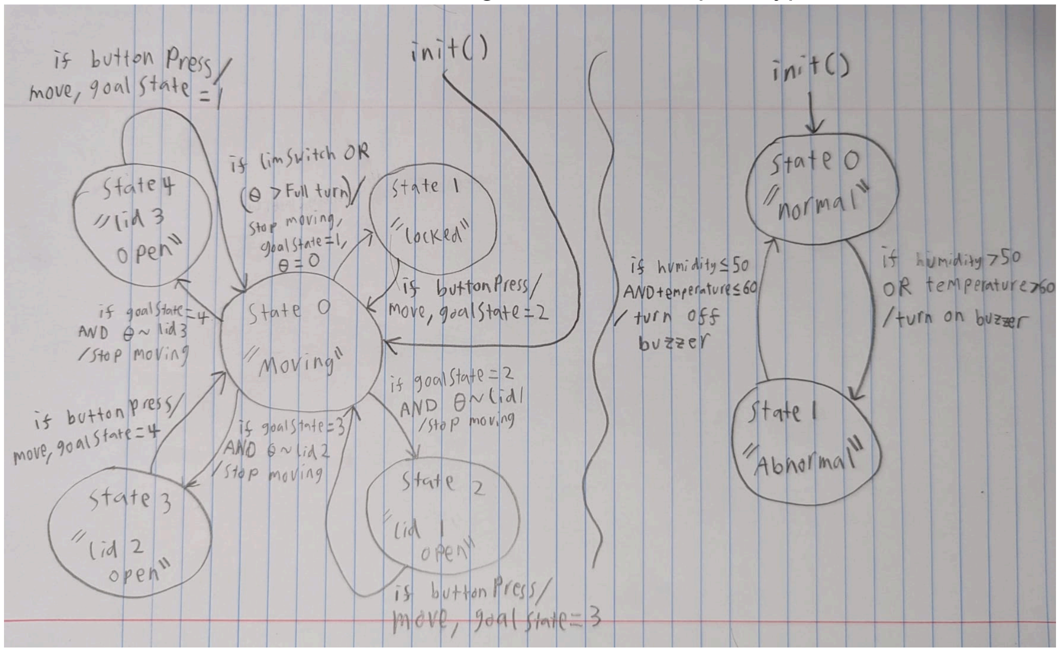


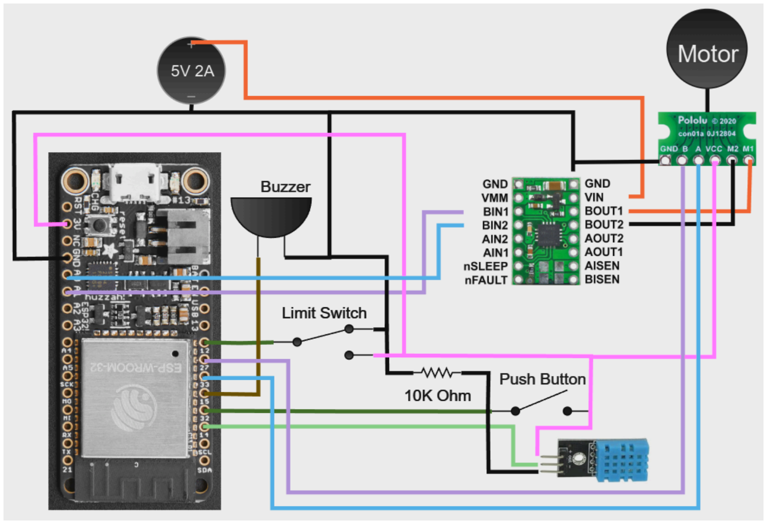




Circuit & State Transition Diagrams

Below is the state diagram of our device, demonstrating the behavior implemented in the final version of the code, as well as the circuit diagram of the final prototype.


init()
if button Press/ move, goal State = 1
State 4 "lid 3 open"
if limSwitch OR (θ > Full turn)/ Stop moving, goal State = 1, θ = 0
State 1 "locked"
if goalState = 4 AND θ ~ lid 3 / Stop moving
State 0 "Moving"
if button Press/ move, goalState = 2
if button press/ move, goalState = 4
if goalState = 3 AND θ ~ lid 2 / Stop moving
if goalState = 2 AND θ ~ lid 1 / stop moving
State 3 "lid 2 open"
State 2 "lid 1 open"
if button Press/ move, goal State = 3
init()
State 0 "normal"
if humidity ≤ 50 AND temperature ≤ 60 / turn off buzzer
if humidity > 50 OR temperature > 60 / turn on buzzer
State 1 "Abnormal"



Reflection

Our group had really good communication and responsiveness, which helped us distribute tasks, share hardware and prototypes and address issues quickly when they came up. We each made use of our individual skills and resources to work on the project independently or together, whenever possible. Our biggest challenge was dealing with a number of issues that arose during the assembly of the final prototype, one day before the functionality showcase. We had to find and buy new hardware, and debug very unexpected issues related to our delicate physical circuit and arduino firmware. The main takeaways from that experience would be to better plan around deadlines and other influences in the project timeline like thanksgiving break - allowing for a greater margin to test the hardware and avoid late design changes. We also learned the importance of careful wiring and good soldering in a compact device, as well as ease of assembly and disassembly to aid with multiple steps of troubleshooting.

Appendix

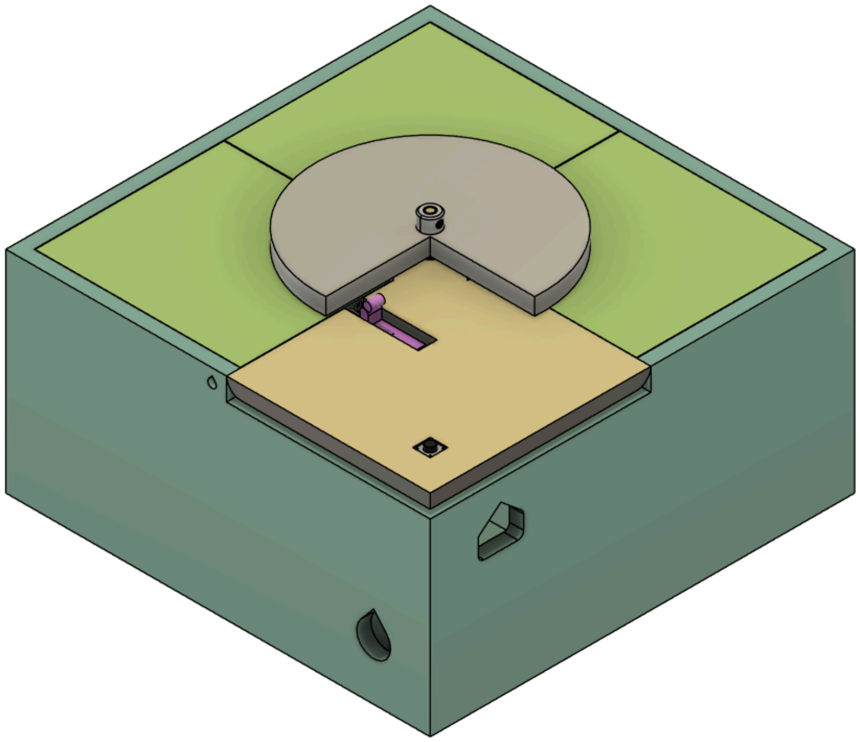
Bill of Materials

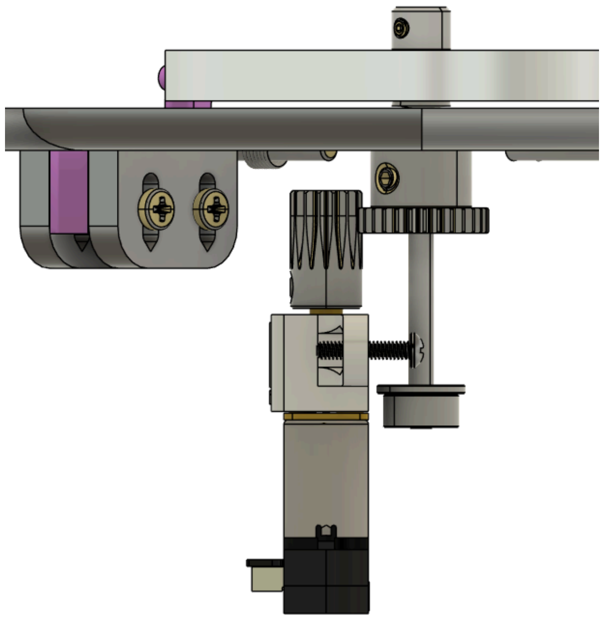
Legend	Mechanical	Parts	Electronic	Custom		
Name	**Part Number**	**Quantity**	**Cost**	**Notes**	**Link**	**CAD / dimensions**
Torsion Spring 180 Degree Left-Hand Wound, 0.186" OD	9271K599	6	$5.76	0.25 in.-lbs. max torque	https://www.mcmaster.com/9271K599/	In purchase link
416 Stainless Steel Dowel Pin 7/64" Diameter, 1" Long	98380A457	6 (2 packs of 5)	$24.50	Two 1in long pins for each lid. One pin for each spring	https://www.mcmaster.com/98380A457/	In purchase link
Rotary Shaft 12L14 Carbon Steel, 3 mm Diameter, 50 mm Long	1327K501	1	$3.83	Disk shaft	https://www.mcmaster.com/1327k501/	In purchase link
Flanged Ball Bearing, Steel, Shielded, Number 603-2Z	57155K546	2	$23.32	Attaches at casing and gearbox	https://www.mcmaster.com/57155K546/	In purchase link
Metal Gear - 20 Degree Pressure Angle Round with Set Screw, 0.5 Module, 15 Teeth, Brass	2664N406	1	$11.81	15 teeth input gear	https://www.mcmaster.com/2664N406/	In purchase link
Metal Gear - 20 Degree Pressure Angle Round with Set Screw, 0.5 Module, 30 Teeth, 3 mm Shaft	2664N424		$17.25	30 teeth output gear	https://www.mcmaster.com/2664N424/	In purchase link
Pololu Micro Metal Gearmotor Bracket Pair - White	1086	1 (pack of 2)	$2.49	Bracket for motor	link	link
18-8 Stainless Steel Spring Lock Washer, Conical, DIN 6796, for M3 Screw Size, 3.2 mm ID, 7 mm OD	91477A121	3 (1 pack of 100)	$5.24	Will replace a shim against the bearings	https://www.mcmaster.com/91477A121/	In purchase link
Carbon Steel Set Screw Collar for 3 mm Shaft Diameter, DIN 705	6056N11	2	$4.00	To hold disk and shaft against bearing	https://www.mcmaster.com/6056N11/	In purchase link
Steel Pan Head Phillips Screw M2.5 x 0.45 mm Thread, 20 mm Long	92005A077	2 (pack of 100)	$6.87	M2.5 screws. For holding the limit switch against the top of the housing	https://www.mcmaster.com/92005A077/	In purchase link
Zinc-Plated Steel Hex Nut Medium-Strength, Class 8, M2.5 x 0.45 mm Thread	90591A270	2 (pack of 100)	$2.22	M2.5 Hex nuts	https://www.mcmaster.com/90591A270/	In purchase link

Group 4 - MEC ENG 102B - Mechatronics Design Fall 2024

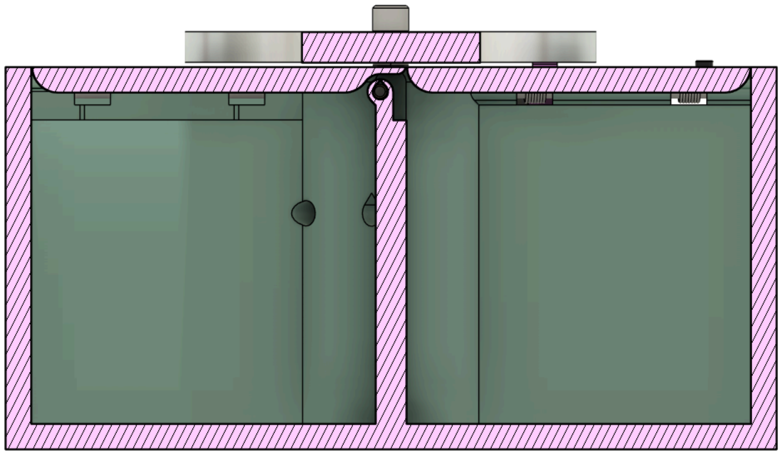
1000:1 Micro Metal Gearmotor HP 6V with 12 CPR Encoder, Side Connector	5177	1	$33.95	At max efficiency: T=2.0kg·cm, ω=26 rpm	link	link
6-Pin Female-Female JST SH-Style Cable 10cm	4765	1	$2.19	Shortest motor cable	link	10cm
Breakout for JST SH-Style Connector, 6-Pin Male Side-Entry (Individually Packaged)	4772	1	$1.25	Breakout board for motor cable	https://www.pololu.com/product/4772	link
Limit switch	-	1 (pack of 10)	$5.99	To detect wedge in disk from underneath	link	Dimensions on amazon's images
Temperature and Humidity Sensor	-	1 (pack of 3)	$5.99		link	incomplete dims here
ESP32 Microcontroller	-	1	$0.00	From microkit	link	See microkit
Mini Speaker	-	1	$0.00	From microkit	link	See microkit
Mini Protoboard	-	1 (pack of 6)	$8.49	To hold esp32 and solder directly to wires	link	Dimensions on amazon's images
10k ohm resistor	-	1	$0.00	From lab		
Tactile push button	-	1	$0.00	From lab/microkit	https://microkit.berkeley.edu/pushbutton-switch/	
Jumper cables	-	-	$0.00	From microkit		
Dual Motor Driver Carrier	2130	1	$9.95		https://www.pololu.com/product/2130	
Enclosure	-	1 (~150g)	$1.50	3D Printed		
Lid	-	3 (~30g)	$0.30	3D Printed		
Electronics cover	-	1 (~10g)	$0.10	3D Printed, covers electronics and holds limit switch		
Disk with wedge	-	1 (~15g)	$0.15	3D Printed		
		Total Cost	**$177.15**			

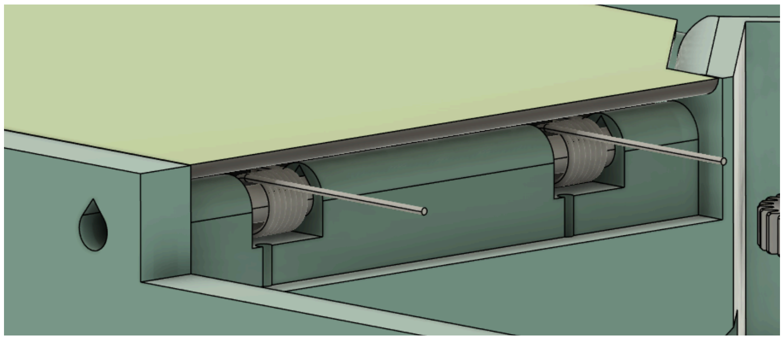
CAD Images

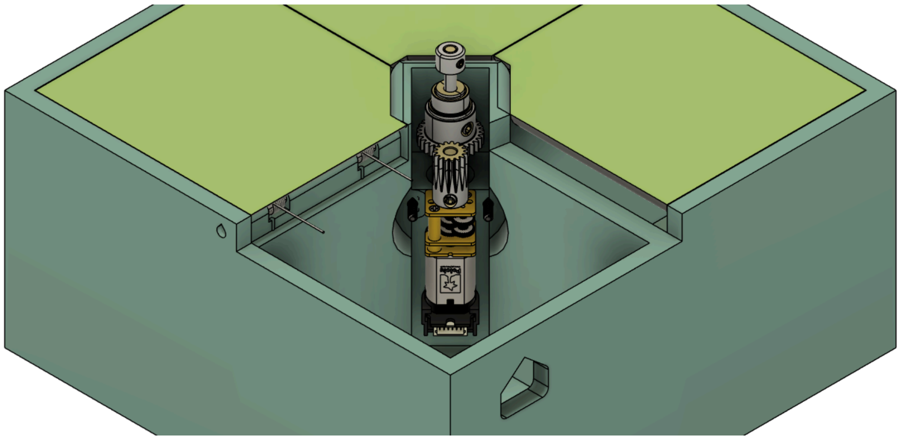


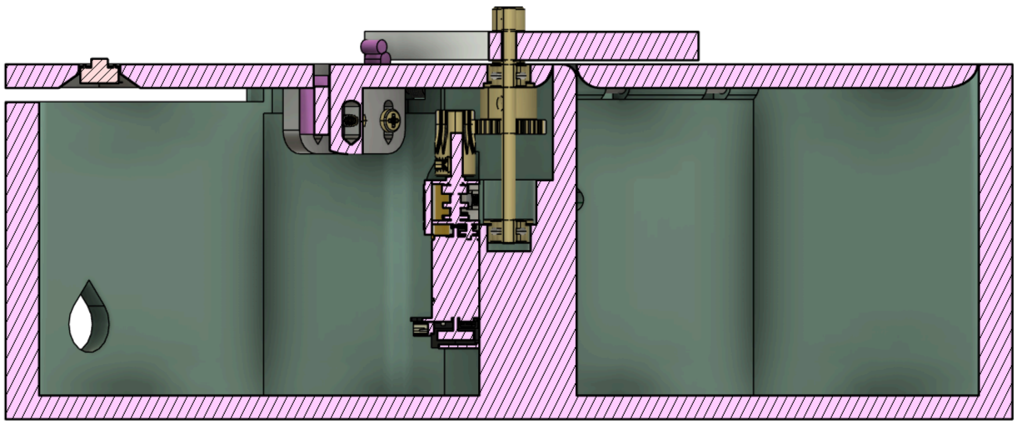


Group 4 - MEC ENG 102B - Mechatronics Design Fall 2024









Code

```
#include <ESP32Encoder.h>
//setup pins
#define BIN_1 25
#define Buzz 15
#define BIN_2 26
#define LED_PIN 13
#define lim 12
#define but 32

// humidity sensor
#include "DHT.h"
#define DPIN 14      //Pin to connect DHT sensor (GPIO number)
#define DTYPE DHT11   // Define DHT 11 or DHT22 sensor type
DHT dht(DPIN,DTYPE);

// encoder library
ESP32Encoder encoder;

//Initialize variables
int theta = 0; //position
int omegaSpeed = 0; //speed
int omegaDes = 0; //desired speed
int omegaMax = 20; //max speed
int D = 0; //motor command
int dir = 1; //direction of spin
int e = 0; //postion error
int ediff = 0; //differencial error
int eprev=0; //previous loop error
float tc = 0; //temperature in C
float tf = 0; //temperature in F
float hu = 0; //humidity in %
int Kp = 20;  //proportional control coefficient
int Kd = 5;  //differencial control coefficient

//Setup interrupt variables ----------------------------
volatile int count = 0;                   // encoder count
volatile bool interruptCounter = false;  // check timer interrupt 1
volatile bool deltaT = false;            // check timer interrupt 2
int totalInterrupts = 0;                 // counts the number of triggering of
the alarm
hw_timer_t* timer0 = NULL;
hw_timer_t* timer1 = NULL;
hw_timer_t* limTimer = NULL;
hw_timer_t* butTimer = NULL;
portMUX_TYPE timerMux0 = portMUX_INITIALIZER_UNLOCKED;
portMUX_TYPE timerMux1 = portMUX_INITIALIZER_UNLOCKED;
portMUX_TYPE timerMuxLim = portMUX_INITIALIZER_UNLOCKED;
portMUX_TYPE timerMuxBut = portMUX_INITIALIZER_UNLOCKED;

// setting PWM properties ---------------------------
const int freq = 5000;
const int ledChannel_1 = 1;
const int ledChannel_2 = 2;
const int resolution = 8;
const int MAX_PWM_VOLTAGE = 255;
const int NOM_PWM_VOLTAGE = 150;

//flags
volatile bool LimTimerDone = true;
volatile bool LimSwitchOn = false;
volatile bool butTimerDone = true;
volatile bool butSwitchOn = false;

//State machine
int fullTurnCount = 12000; //encoder counts per full turn of output (6
counts/turn * 1000 motor gearing ratio * 2 shaft gearing ratio)
byte state = 0; //Initialize state
byte goalState = 1; //Initialize goal state
const double tolerance = 20; //positional tolerance
const double offset = 250;
int lid1 = (fullTurnCount/4)+offset;
int lid2 = (fullTurnCount/2)+offset;
int lid3 = (3*fullTurnCount/4)+offset;
String desiredCompartment = " ";
char condition = 'n'; //state variable for humidity sensor
```

```
//Initialization ------------------------------------
void IRAM_ATTR onTime0() { //timer0 isr
  portENTER_CRITICAL_ISR(&timerMux0);
  interruptCounter = true;
  portEXIT_CRITICAL_ISR(&timerMux0);
}

void IRAM_ATTR onTime1() { //timer1 isr
  portENTER_CRITICAL_ISR(&timerMux1);
  count = encoder.getCount(); //get encoder count over this period dt
  encoder.clearCount();
  deltaT = true;
  portEXIT_CRITICAL_ISR(&timerMux1);
}

void IRAM_ATTR isr() {  //limit switch isr
  if (LimTimerDone) {
    LimSwitchOn = true;
    LimTimerDone = false;
    timerWrite(limTimer, 0);
    timerStart(limTimer);
  };
}

void IRAM_ATTR isrBut() {  //button isr
  if (butTimerDone) {
    butSwitchOn = true;
    butTimerDone = false;
    timerWrite(butTimer, 0);
    timerStart(butTimer);
  };
}

void IRAM_ATTR onTime() { //limit switch debounce timer isr
  portENTER_CRITICAL_ISR(&timerMuxLim);
  timerStop(limTimer);
  LimTimerDone = true;
  portEXIT_CRITICAL_ISR(&timerMuxLim);
}

void IRAM_ATTR onTimeBut() { //button debounce timer isr
  portENTER_CRITICAL_ISR(&timerMuxBut);
  timerStop(butTimer);
  butTimerDone = true;
  portEXIT_CRITICAL_ISR(&timerMuxBut);
}

void setup() { //SETUP
  dht.begin(); // for humidity sensor
  //set up pins
  pinMode(LED_PIN, OUTPUT);
  pinMode(Buzz, OUTPUT);
  digitalWrite(LED_PIN, LOW);  // sets the initial state of LED as turned-off

  Serial.begin(115200);
  ESP32Encoder::useInternalWeakPullResistors = puType::up;  // Enable the weak
pull up resistors
  encoder.attachHalfQuad(33, 27);                           // Attache pins for
use as encoder pins
  encoder.setCount(0);                                      // set starting
count value after attaching

  ledcAttach(BIN_1, freq, resolution);
  ledcAttach(BIN_2, freq, resolution);

  // initialize timers
  timer0 = timerBegin(1000000);              // Set timer frequency to 1Mhz
  timerAttachInterrupt(timer0, &onTime0);  // Attach onTimer0 function to our
timer.
  timerAlarm(timer0, 5000000, true, 0);     // 5000000 * 1 us = 5 s, autoreload
true

  timer1 = timerBegin(1000000);              // Set timer frequency to 1Mhz
  timerAttachInterrupt(timer1, &onTime1);  // Attach onTimer1 function to our
timer.
```

```
  timerAlarm(timer1, 10000, true, 0);      // 10000 * 1 us = 10 ms, autoreload
true

  pinMode(lim, INPUT);                // configures the limit switch pin as an
input
  attachInterrupt(lim, isr, FALLING);  // set the "lim" pin as the interrupt
pin; call function named "isr" when the interrupt is triggered; "FALLING" means
triggering interrupt when the pin goes from HIGH to LOW

  limTimer = timerBegin(1000000);
  timerAttachInterrupt(limTimer, &onTime);
  timerAlarm(limTimer, 50000, true, 0);  //setup limit switch debounce time

  // for button
  pinMode(but, INPUT);                // configures the specified button pin as
an input
  attachInterrupt(but, isrBut, RISING); // set the "but" pin as the interrupt
pin; call function named "isrBut" when the interrupt is triggered; "RISING"
means triggering interrupt when the pin goes from LOW to HIGH

  butTimer = timerBegin(1000000);
  timerAttachInterrupt(butTimer, &onTimeBut);
  timerAlarm(butTimer, 200000, true, 0);  //setup button debounce time
}

void loop() { //MAIN LOOP

//must check for limit switch and button in every loop regardless of state.
Leaving the function calls here reduced duplicate statements.
  if (CheckForLimSwitch()) {
    LimResponse();
  }
  if (CheckForButSwitch()) {
    butResponse();
  }

  switch (state) {
    case 0:  //moving state - reached upon each button press
```

```
      omegaDes = omegaMax; //we're moving
      //depending on our goal state, we will transition to different states at
different times.
      if ((goalState==2)&&(abs(theta-lid1)<tolerance)) { //if you reach the
goal state 2, switch state to 2
        state = 2;
      }
      else if ((goalState==3)&&(abs(theta-lid2)<tolerance)) { //if you reach
the goal state 3, switch state to 2
        state = 3;
      }
      else if ((goalState==4)&&(abs(theta-lid3)<tolerance)) { //if you reach
the goal state 4, switch state to 2
        state = 4;
      }
      break;

    case 1:   // the locked state. Is reached after limit switch is hit or a
full turn count is reached
      omegaDes = 0; //stop moving
      break;

    case 2:
      omegaDes = 0; //stop moving
      break;

    case 3:
      omegaDes = 0; //stop moving
      break;

    case 4:
      omegaDes = 0; //stop moving
      break;
  }


  if (deltaT) { //running a loop every deltaT, not every loop.
    portENTER_CRITICAL(&timerMux1);
    deltaT = false;
    portEXIT_CRITICAL(&timerMux1);

    // dht sensor
    tc = dht.readTemperature(false);  //Read temperature in C
    tf = dht.readTemperature(true);   //Read Temperature in F
    hu = dht.readHumidity();          //Read Humidity

  switch (condition) { //Switch state for sensor/buzzer behavior

    case 'n': // normal condition
      if (checkForAbnormalConditions(hu, tc)){
        abnormalConditionsResponse();
        condition = 'a';
      }
      break;

    case 'a': // abnormal condition
      if (!checkForAbnormalConditions(hu, tc)){
      condition = 'n';
      normalConditionsResponse();
      }
      break;
  }

    // update position
    theta += count;

    // speed controller
    omegaSpeed = count; //read current speed
    e = omegaDes - omegaSpeed; //find error
    eprev=e; //save current speed
    ediff=e-eprev; //calculate differencial error
    D = Kp*e+Kd*ediff; //PD controller

    //Ensure that you don't go past the maximum possible command
    if (D > MAX_PWM_VOLTAGE) {
      D = MAX_PWM_VOLTAGE;
    } else if (D < -MAX_PWM_VOLTAGE) {
      D = -MAX_PWM_VOLTAGE;
    }

    //write motor commands
    if (D > 0) {
      ledcWrite(BIN_1, LOW);
      ledcWrite(BIN_2, D);
    } else if (D < 0) {
      ledcWrite(BIN_2, LOW);
      ledcWrite(BIN_1, -D);
    } else {
      ledcWrite(BIN_2, LOW);
      ledcWrite(BIN_1, LOW);
    }

    plotControlData(); //printing for debugging
  }
}

bool CheckForLimSwitch() {
  if (theta>fullTurnCount) { //after a full turn, emulate a limit switch input
(increases robustness against failed limit switch signals)
    return true;
  }
  else{
    return LimSwitchOn; //returns limit switch flag from isr
  }
}

void LimResponse() {
  LimSwitchOn = false; //reset flag
  theta = 0; //reset position
  state = 1; // set state to all closed
  goalState = 1;
  Serial.println("LIMIT SWITCH RESPONSE");
}
```

```
bool CheckForButSwitch() {
  return butSwitchOn; //returns button flag from isr
}

void butResponse() {
  butSwitchOn = false; //reset flag
  state = 0; //switch to the moving state
  if (goalState==4){ //increment goal state 1>2>3>4>1
    goalState=1;
  }
  else{
    ++goalState;
  }
  Serial.println("BUTTON RESPONSE");
}

bool checkForAbnormalConditions(float hu, float temp){
  if ((hu > 50.0) || (temp > 60)){ //check if humidity and temperature are out of acceptable range
    return true;
  }
  return false;
}

void abnormalConditionsResponse(){
  tone(Buzz, 640);
  Serial.println("abnormal conditions: turn on buzzer");
  digitalWrite(LED_PIN, HIGH);
}

void normalConditionsResponse(){
  Serial.println("normal conditions: turn off buzzer");
  noTone(Buzz);
  digitalWrite(LED_PIN, LOW);
}
```

```
void plotControlData() { //for debugging
  Serial.print("temp:");
  Serial.print(tc);
  Serial.print(" ");
  Serial.print("humidity:");
  Serial.print(hu);
  Serial.print(" ");
  Serial.print("Position:");
  Serial.print(theta);
  Serial.print(" ");
  Serial.print("D:");
  Serial.print(D);
  Serial.print(" ");
  Serial.print("state:");
  Serial.print(state);
  Serial.print(" ");
  Serial.print(" goalState:");
  Serial.print(goalState);
  Serial.print(" ");
  Serial.print("omega des:");
  Serial.print(omegaDes);
  Serial.print(" ");
  Serial.print(" count or velocity:");
  Serial.println(count);
}
```